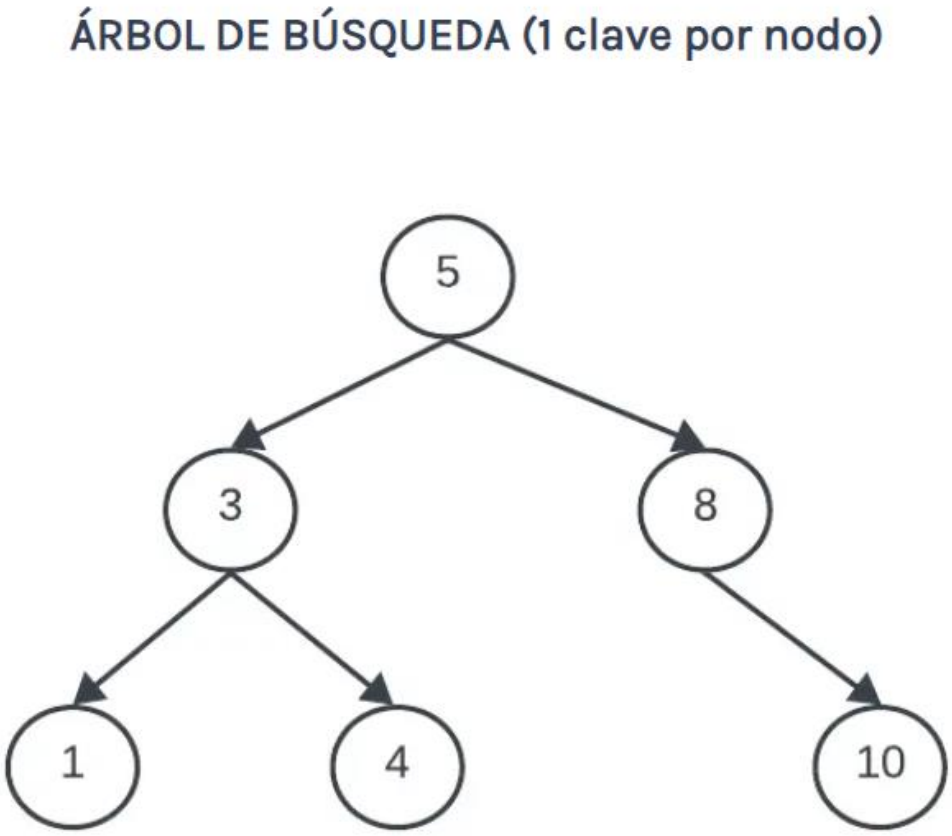
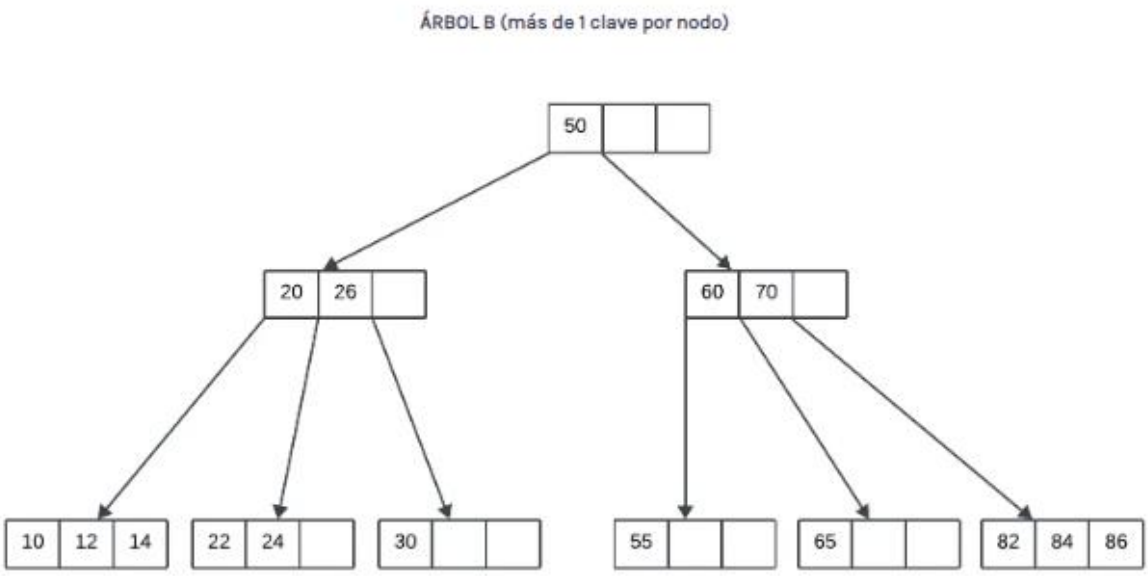
¿Qué son los árboles B?

Los árboles B, tal y como indica su nombre, son una estructura de datos arborescente. El árbol B es un árbol de búsqueda, por lo tanto, cada nodo está asociado a una clave. ¿Cuál es su particularidad? Pues que en esta estructura los nodos tienen asociados más de una clave.



ÁRBOL B (más de 1 clave por nodo)

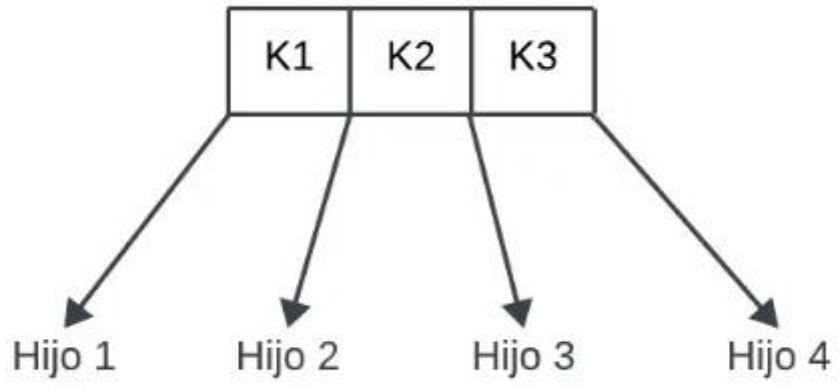


En el árbol binario de búsqueda los hijos derechos de un nodo contienen las claves más grandes que la clave del nodo padre y, a su vez, los hijos izquierdos contienen las claves más pequeñas que la clave del nodo padre. Algo similar ocurre con el árbol B

Estructura

El esqueleto de esta estructura de datos. Primero de todo, debemos conocer un concepto importante, el orden:

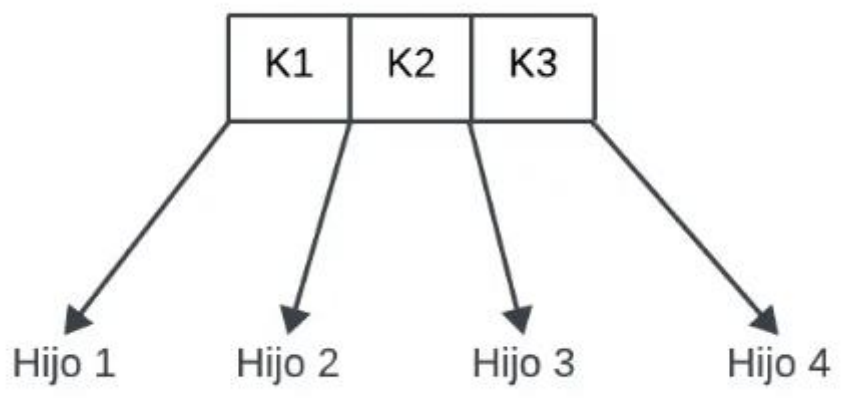
- **Orden:** número máximo de hijos que puede tener un nodo (lo denotaremos como «m»).
  A partir del orden podemos saber el número máximo de claves que puede tener un nodo. Este siempre será m – 1, ejemplo.



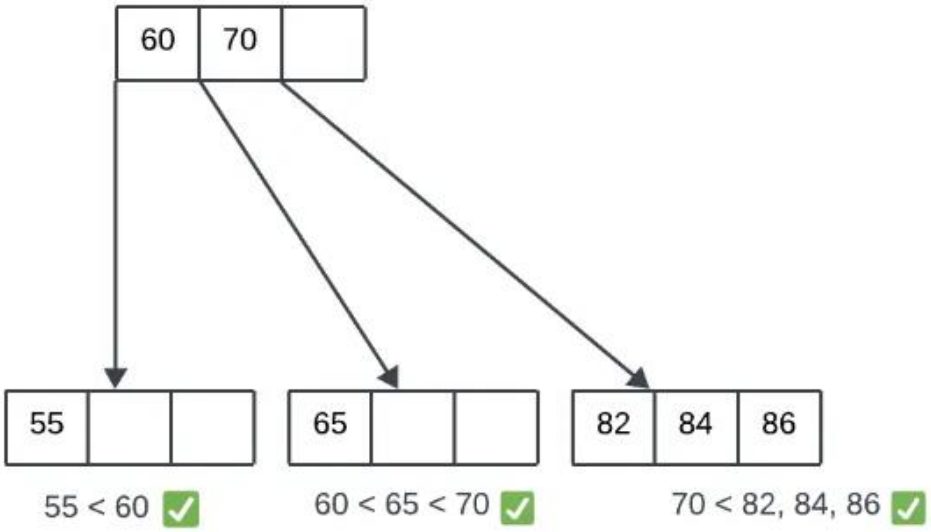
Comprobar que, para un nodo de orden 4 (puede tener máximo 4 hijos), efectivamente el número de claves es m – 1 = 4 – 1 = **3**.

La parte más importante de todas, ¿los árboles B hacían una cosa parecida a los árboles de búsqueda a la hora de organizar los nodos por claves?

Para el nodo anterior:



- Las claves están ordenadas de forma ascendente, es decir, K1 < K2 < K3
- El hijo 1 contendrá claves más pequeñas que K1
- El hijo 2 contendrá claves más grandes que K1 y más pequeñas que K2
- El hijo 3 contendrá claves más grandes que K2 y más pequeñas que K3
- El hijo 4 contendrá claves más grandes que K3
  Analizar el nodo del ejemplo para ver si se cumple esta estructura.



Propiedades de los árboles B

Para poder conseguir tener un árbol B, no solo debemos comprobar que esté definido con la estructura anterior ya que también deben cumplirse una serie de propiedades fundamentales:

PROPIEDAD 1: los árboles B son SIEMPRE balanceados.

PROPIEDAD 2: todos los nodos de un árbol de orden m excepto la raíz deben cumplir que

$$\text{número mínimo de claves} = \lceil \frac{m}{2} \rceil - 1$$

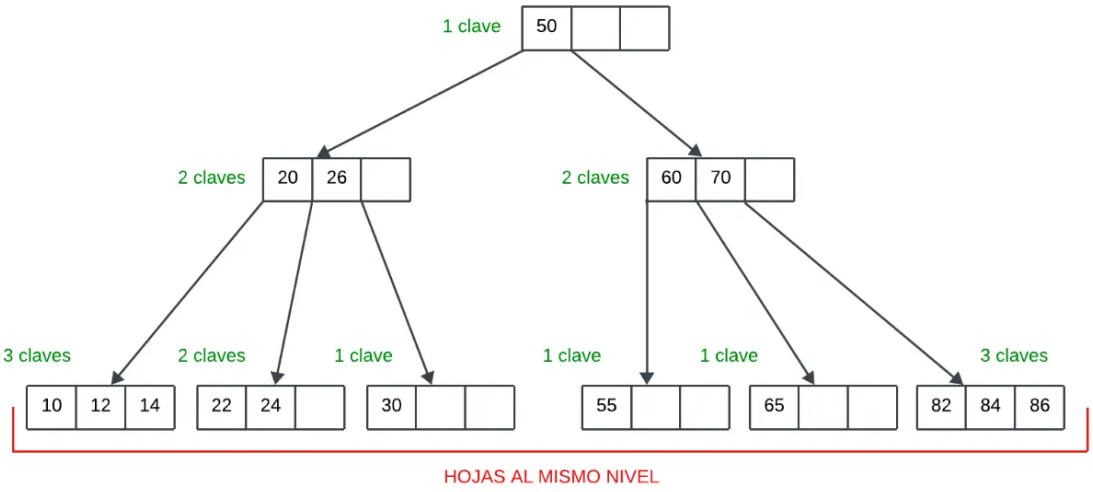
Este símbolo ⌈ ⌉ significa redondear hacia arriba, por ejemplo:

$$\lceil \frac{5}{2} \rceil = 3$$

PROPIEDAD 3: todas las hojas, es decir, los nodos sin hijos, están al mismo nivel.

PROPIEDAD 4: todo nodo interno (no es hoja) con n claves tiene n + 1 hijos.

Anteriormente, se comprobó que el árbol B ejemplo cumplía con la estructura. Veamos si cumple con estas tres propiedades.



Verificamos que:

- Las hojas están al mismo nivel ✓
- Está equilibrado (lo implica que todas la hojas estén al mismo nivel) ✓
- Todos los nodos excepto la raíz tienen un mínimo de 1 clave ✓
  $\lceil \frac{m}{2} \rceil - 1 = \lceil \frac{4}{2} \rceil - 1 = 1$
- Todos los nodos internos tienen n + 1 hijos, donde n es el número de claves ✓

Algoritmos de inserción y eliminación

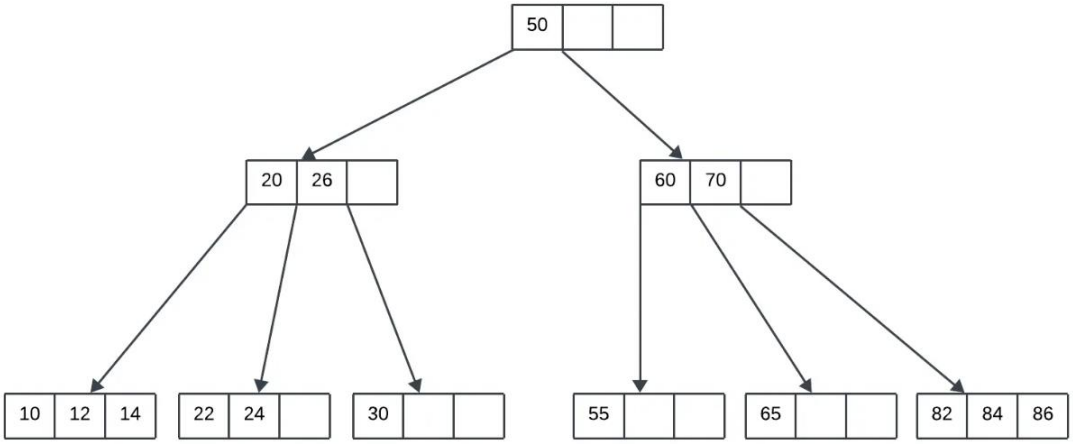
Esta estructura de datos mediante algoritmos determinados, se puede insertar y eliminar claves sin alterar su estructura ni propiedades fundamentales. Cómo insertar una nueva clave en un árbol B.

Inserción en un árbol B

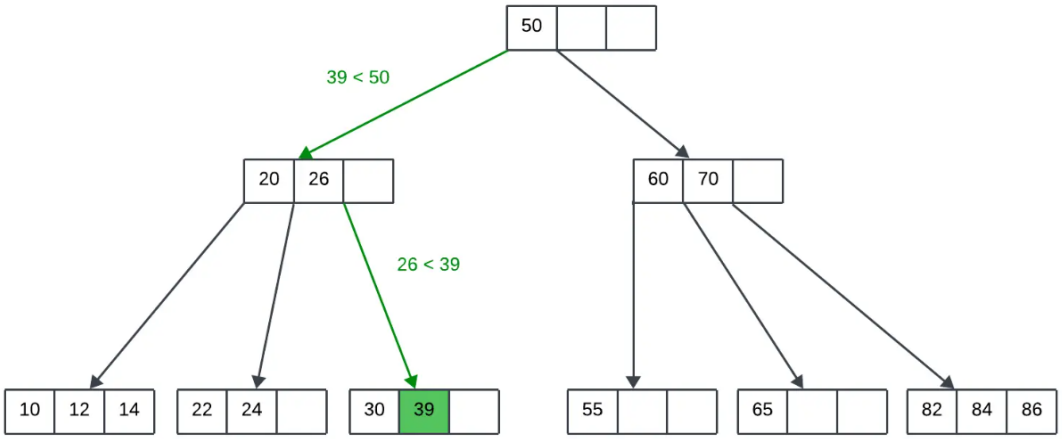
Al insertar una nueva clave solo debemos introducirla en la posición que le corresponda. No obstante, podemos encontrarnos con algunas dificultades a la hora de añadirla.

Para ello, operar sobre este árbol:

Caso 1: la clave puede insertarse sin problemas

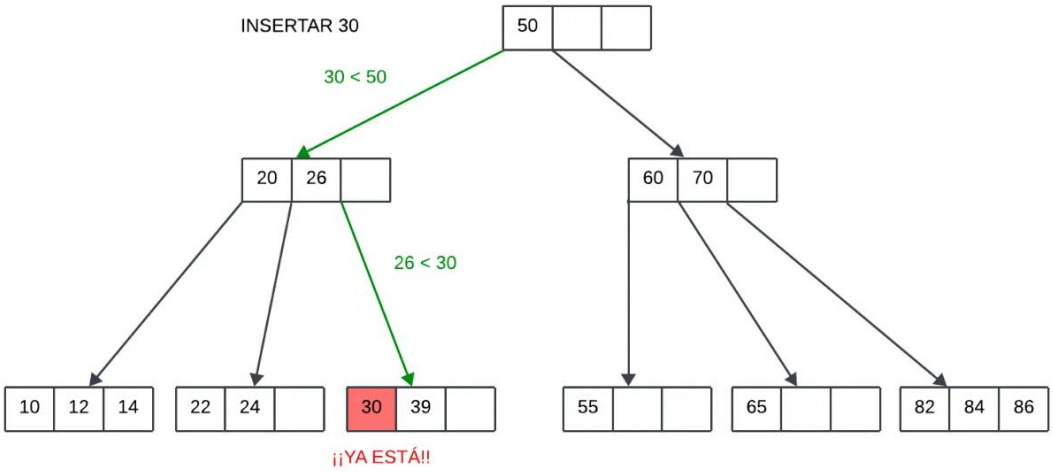


Empezar por el más fácil. Añadir el nodo 39.



Caso 2: La clave ya se encuentra en el árbol

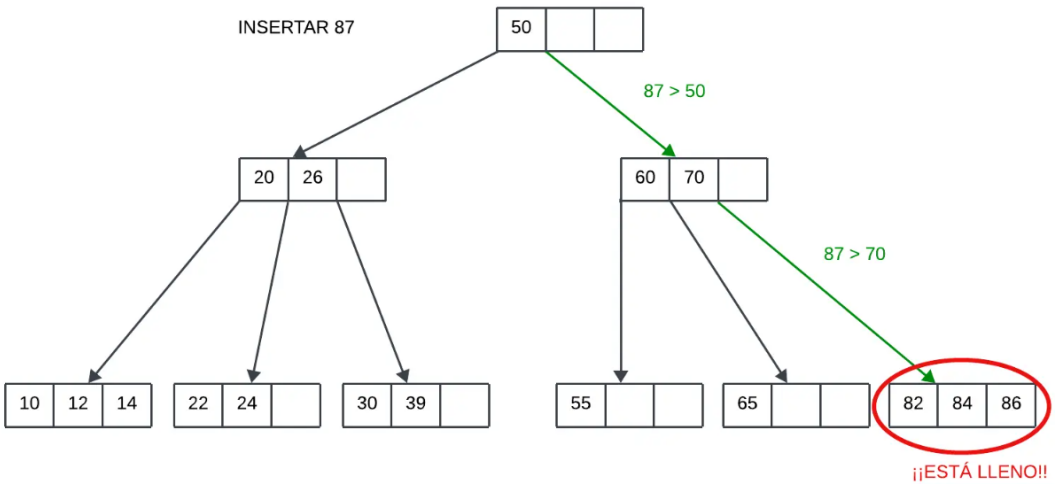
En este caso la estructura del árbol quedaría exactamente igual, no tenemos que cambiar nada.



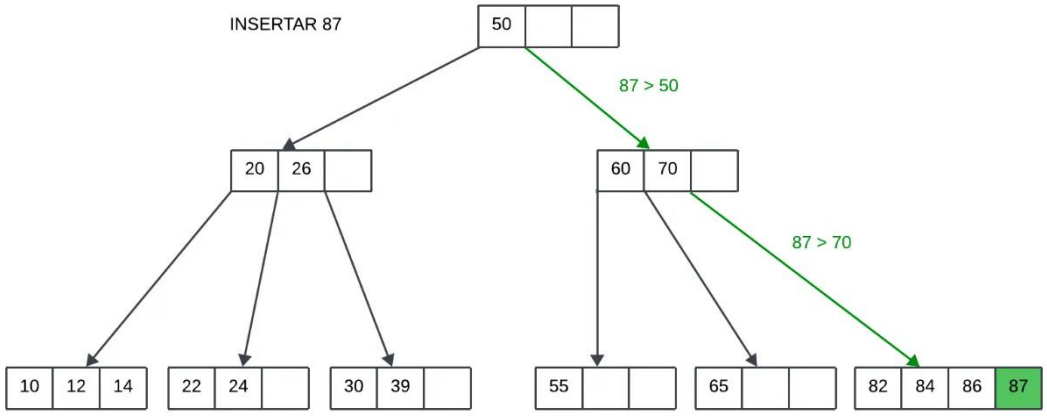
Eso sí, hay que pensar que los árboles B sirven para almacenar datos de forma ordenada y que las claves solo son un identificador de estos datos. Es decir, este árbol sirve para guardar los nombres de los alumnos de un instituto. La clave 30 tiene asociada, por ejemplo, ‘Juan’. Si insertásemos la clave 30 con un nuevo nombre, por ejemplo, ‘Javier’ el árbol quedaría exactamente igual pero la información asociada a la clave cambiaría.

Caso 3: el nodo está lleno (split)

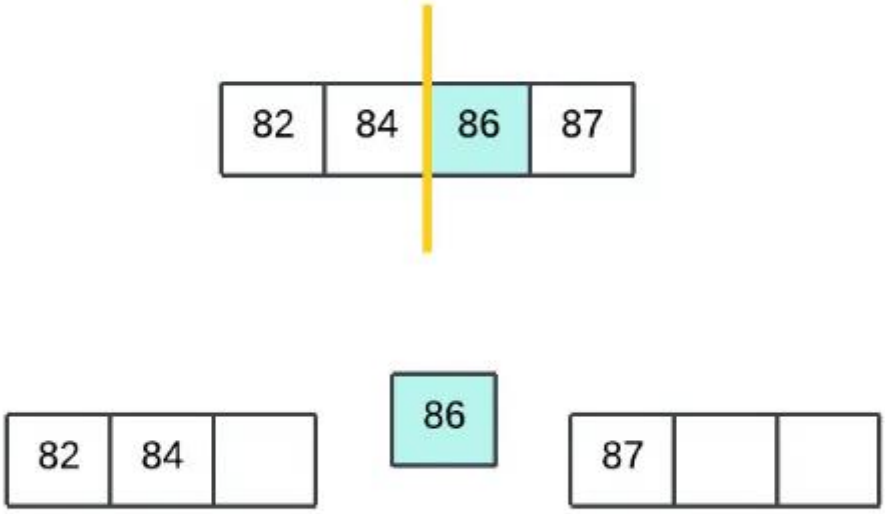
Ahora insertar la clave 87.

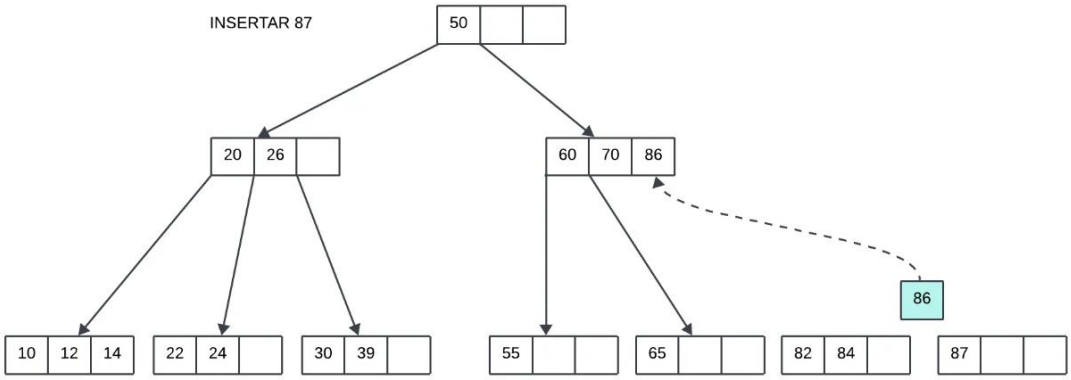


¿Qué pasa? Pues que el nodo al que le toca introducirse está lleno. Para solucionarlo, insertar la clave al nodo lleno como si hubiera un espacio +

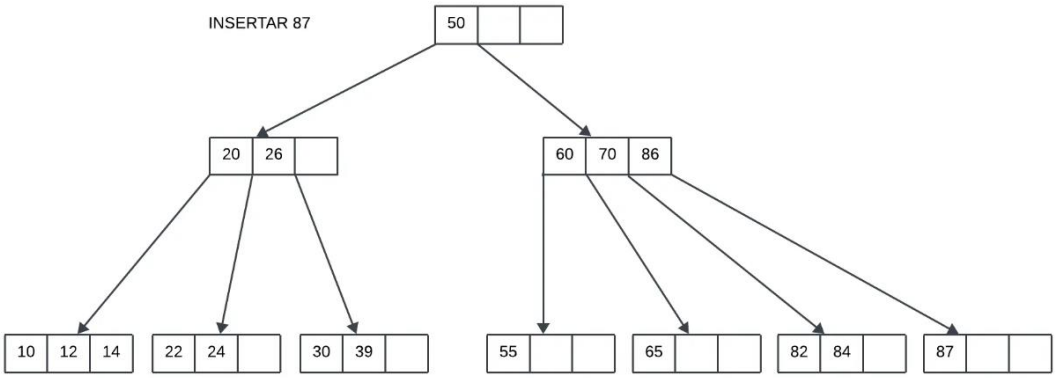


Pero no podemos dejar el árbol así ya que los nodos deben ser de orden 4 y ahora tenemos un nodo de orden 5. Para solucionarlo partiremos este nodo por la mitad (split) y la clave del medio subirá al padre.



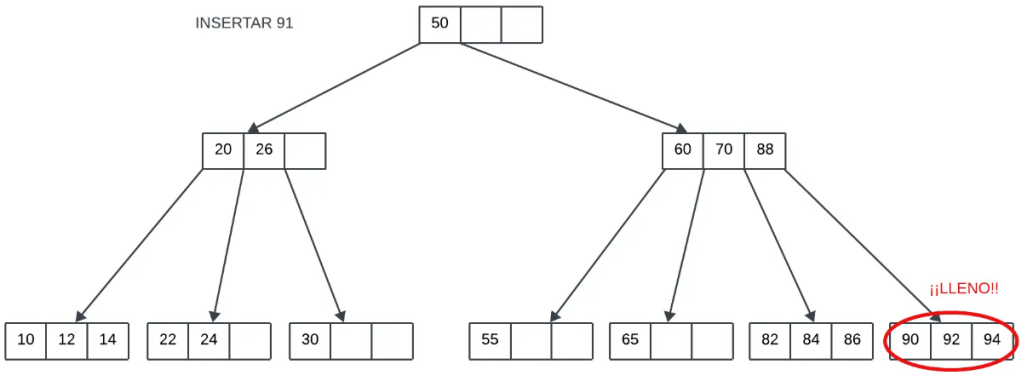


Finalmente, solo nos quedará añadir las conexiones entre el nodo padre y los nuevos hijos respectando la estructura fundamental.

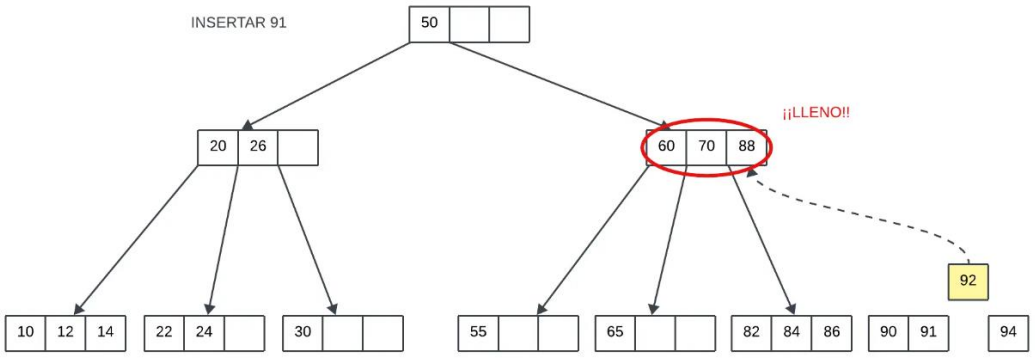


Extra: ¿qué hago si al hacer un split el padre también está lleno?

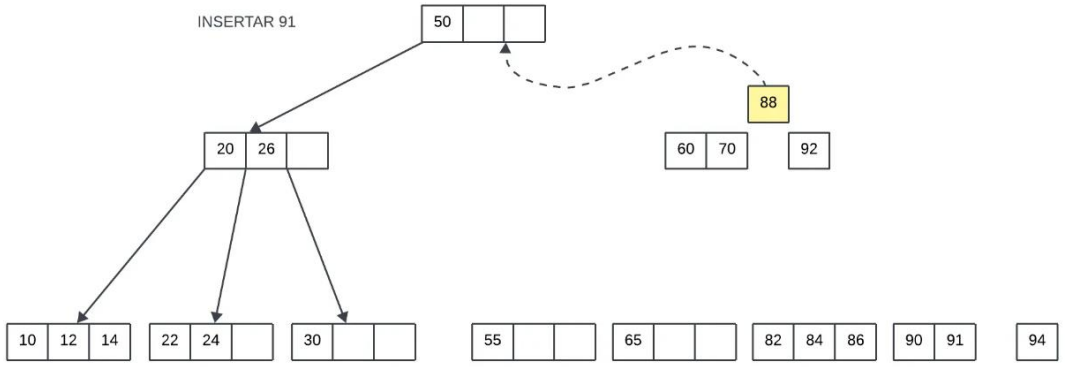
Ahora añadimos el 91 a este árbol. Encontramos el nodo pero está lleno, por lo tanto, hacemos un split.



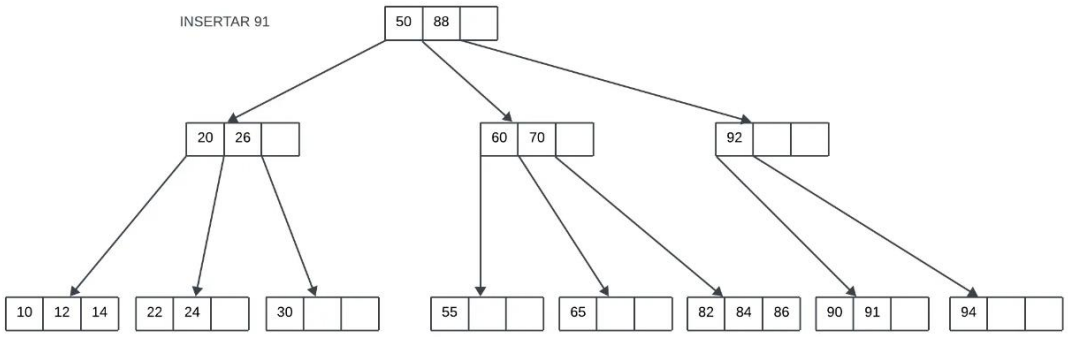
El nodo padre al que debía subir el elemento del medio está lleno también. En consecuencia, nos vemos obligados a hacer otra división.



Finalmente, añadimos las conexiones correspondientes con los nuevos nodos.



Finalmente, añadimos las conexiones correspondientes con los nuevos nodos.



Eliminación en un árbol B

Para eliminar una clave de un nodo debemos mirar si está y, si la encontramos, eliminarla. Pero los nodos de los árboles B deben cumplir la siguiente propiedad:

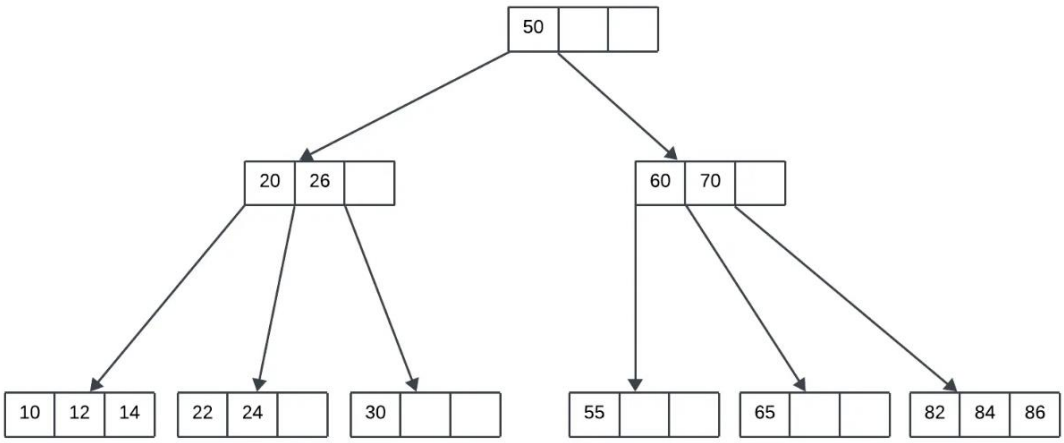
$$\text{número mínimo de claves} = \lceil \frac{m}{2} \rceil - 1$$

Caso 1: la clave no está en el árbol

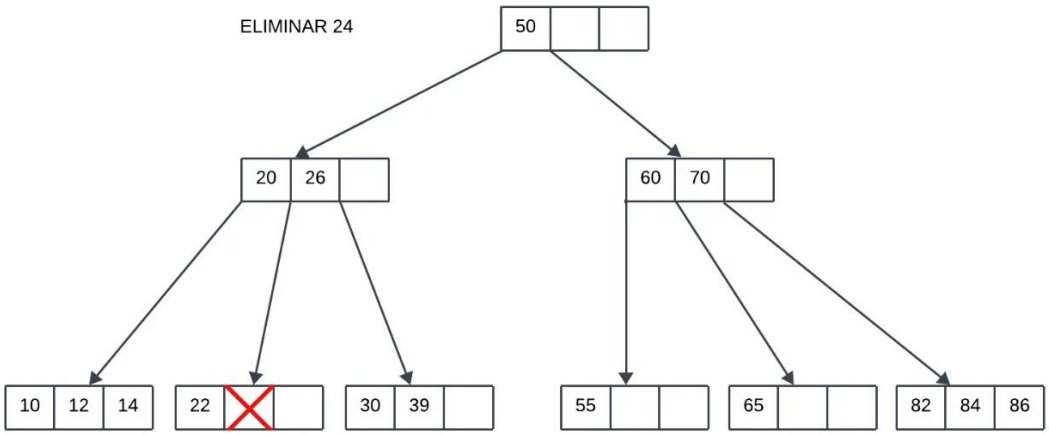
En este caso no debemos hacer nada y dejar el árbol tal y como estaba.

Caso 2: la clave si está en el árbol

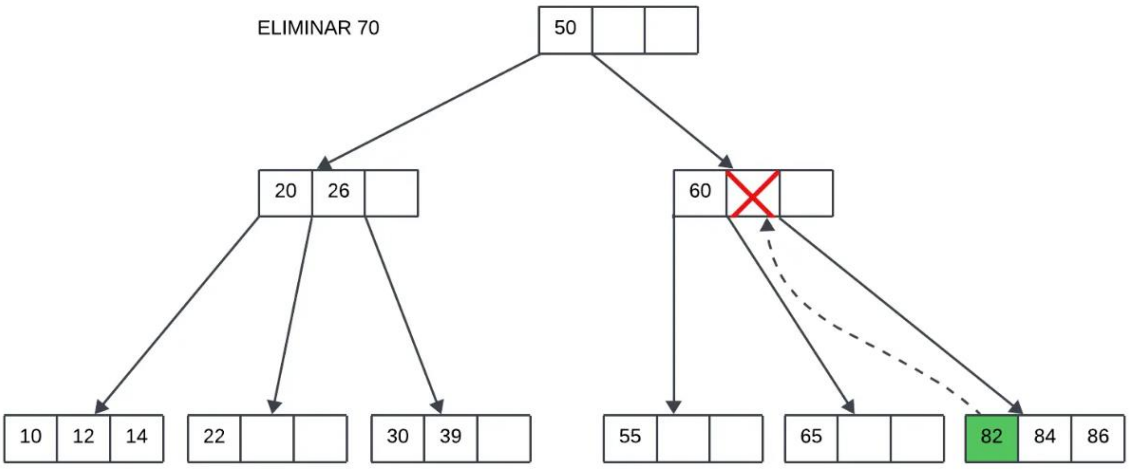
En este caso tenemos dos opciones: que la clave esté en una hoja o que esté en un nodo interno. Si está en una hoja, la eliminaremos y ya habremos acabado. En cambio, si se encuentra en un nodo interno, la eliminaremos y la siguiente clave en orden ascendente la reemplazará. Ejemplo



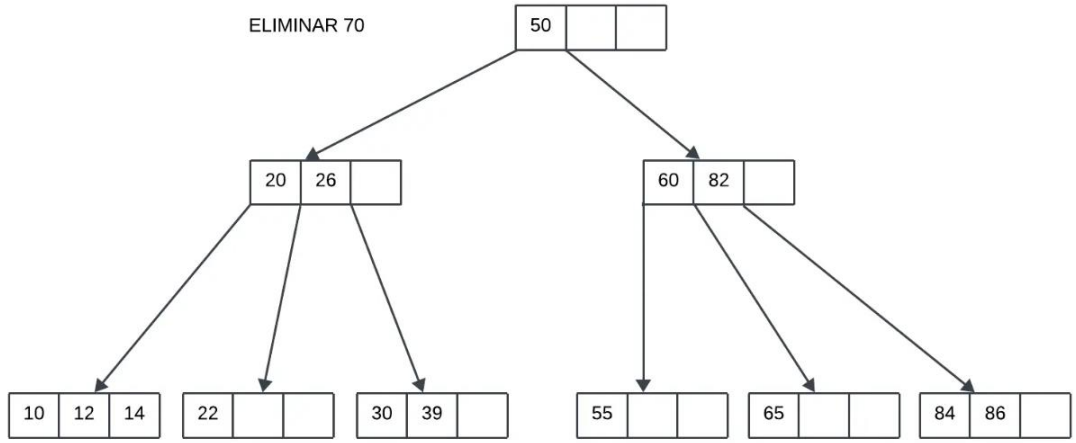
Eliminamos el 24 que está en una hoja.



Si se eliminara el 70, que está en un nodo interno, la siguiente clave en orden ascendente lo reemplazaría. Vemos que esta es 82.



El árbol acabaría así:

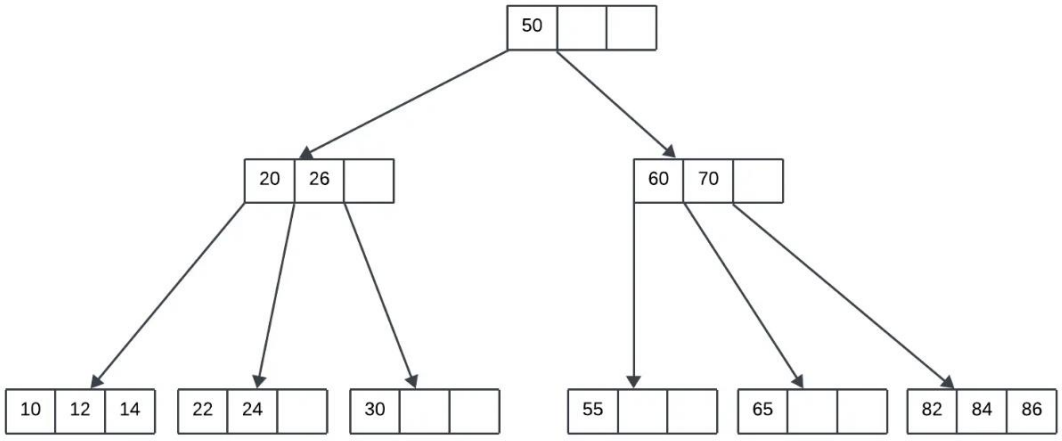


Caso 3: eliminar una clave genera déficit

Si al eliminar una clave generamos déficit en algún nodo, es decir, dejamos un nodo sin el número mínimo de claves tenemos dos opciones:

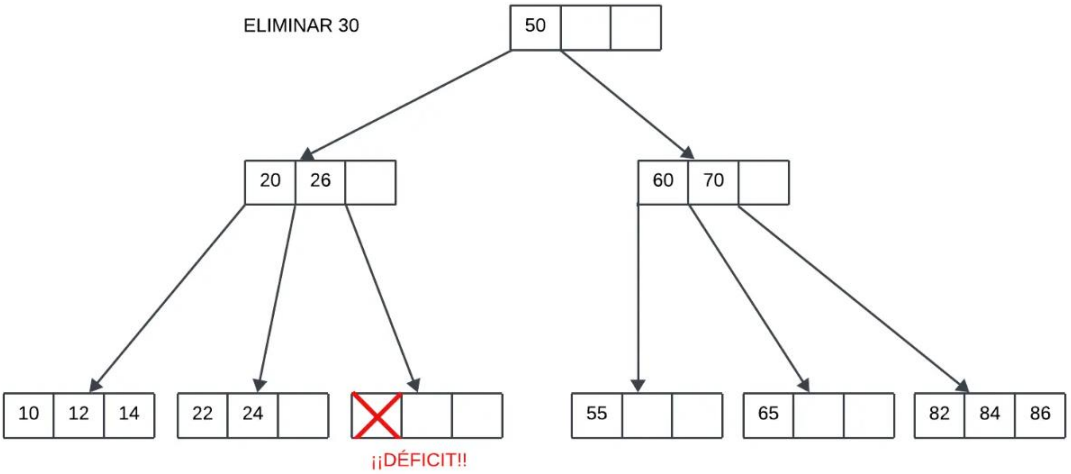
La primera opción será que el hermano izquierdo o el derecho le presten una clave. Para ello, será necesario que tengan más del número mínimo, si no, se quedarían en déficit y seguiríamos con el mismo problema.

La segunda opción será fusionar el nodo con déficit con uno de sus hermanos. Este solo será viable si no se puede aplicar la primera. Veamos un ejemplo para cada una:



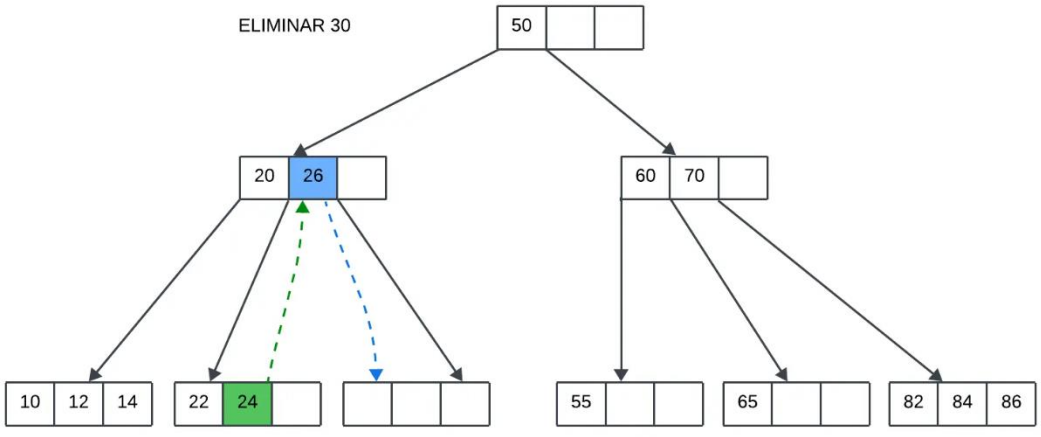
Empezaremos con el primer caso eliminando la clave 30 del siguiente árbol:

Al eliminarla generamos déficit.

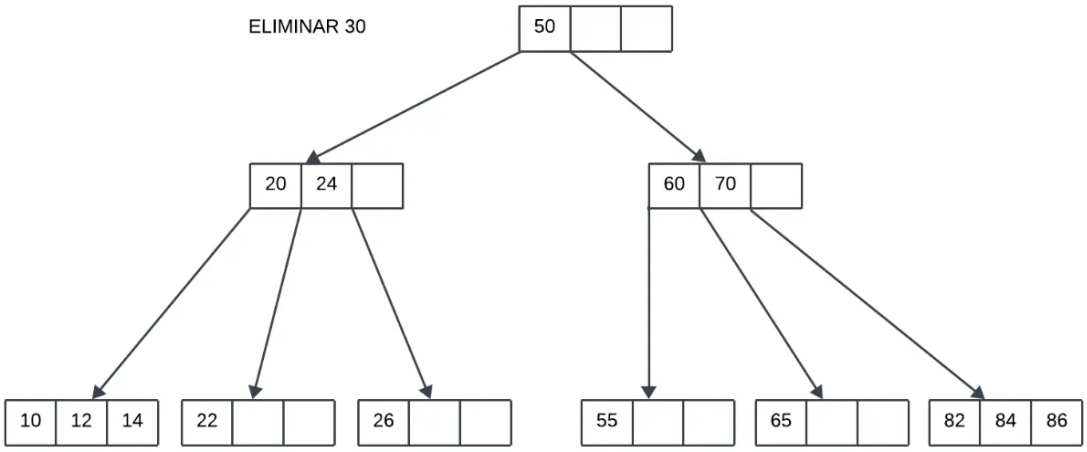


Observamos que el hermano izquierdo puede prestarle una clave.

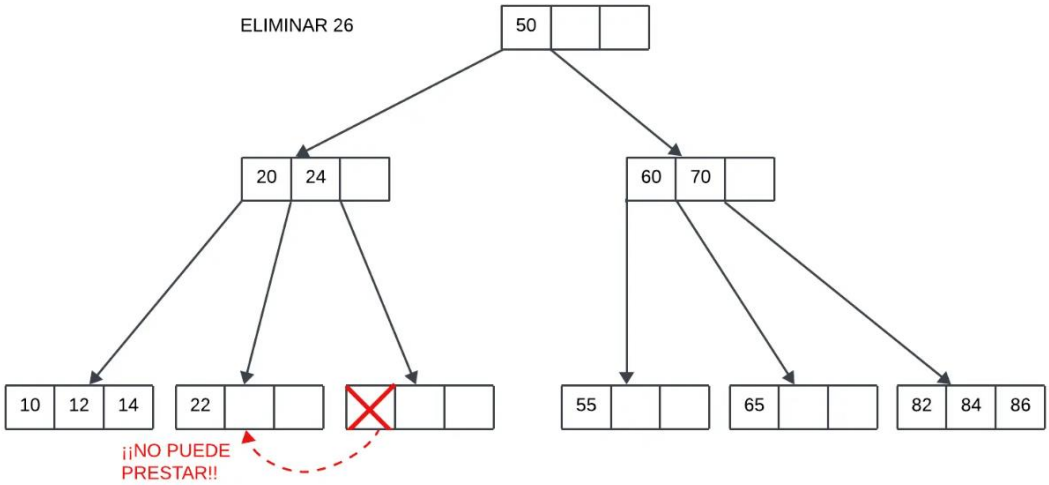
¡IMPORTANTE! Para mantener las propiedades de un árbol B el hermano no le presta una clave directamente. Lo que se hace es subir al padre la clave del hermano (la más grande si es el izquierdo y la más pequeña si es el derecho) y bajamos la clave del padre al nodo con déficit.



Así es como queda finalmente:

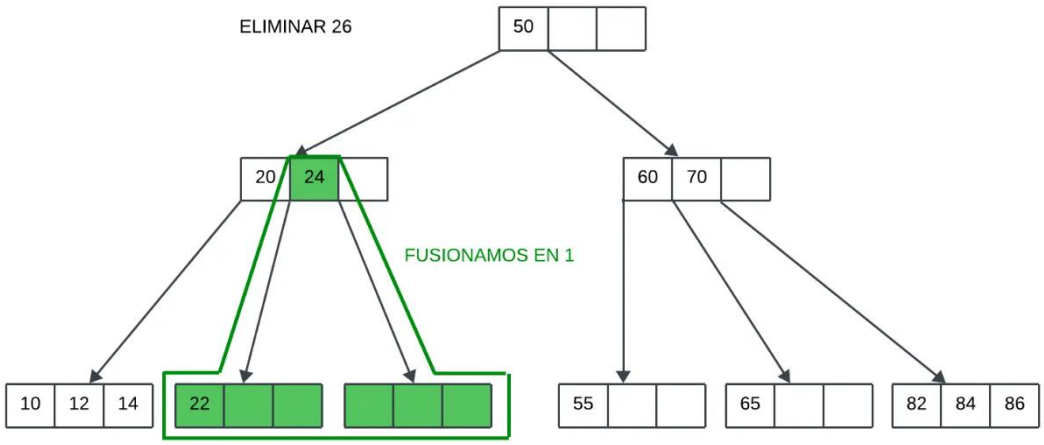


Ahora, sobre este mismo árbol suprimiremos la clave 26. El hermano izquierdo no puede prestarle ninguna clave y, por si no fuera suficiente, no tiene hermano derecho.

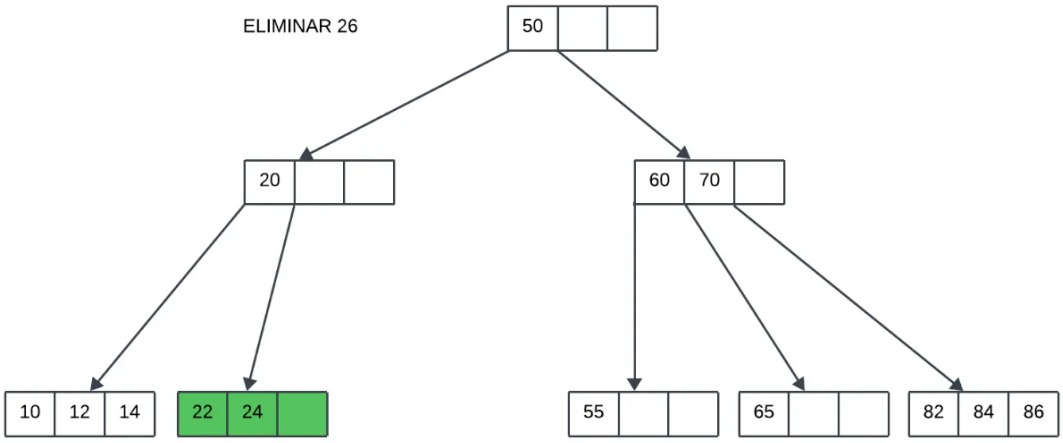


Es en este momento es cuando se debe fusionar (merge) el nodo con déficit con uno de sus hermanos, en este caso, el izquierdo ya que no tiene hermano derecho.

¡IMPORTANTE! En la fusión uniremos las claves del nodo con déficit, las del hermano elegido y la clave del nodo padre de la cual cuelgan estos dos nodos.



Después de todo, terminamos con este árbol:



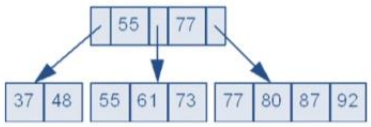
Árboles B+

—Los árboles B+ constituyen otra mejora sobre los árboles B, pues conservan la propiedad de acceso aleatorio rápido y permiten además un recorrido secuencial rápido.

—En un árbol B+ todas las claves se encuentran en hojas, duplicándose en la raíz y nodos interiores aquellas que resulten necesarias para definir los caminos de búsqueda.

Árboles B+

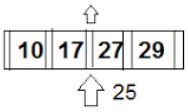
—Para facilitar el recorrido secuencial rápido las hojas se pueden vincular, obteniéndose de esta forma una trayectoria secuencial para recorrer las claves del árbol.



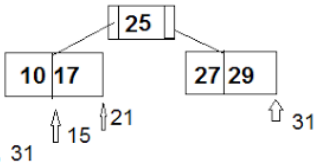
EJEMPLO

Insertar :10, 27, 29, 17,25, 21, 15, 31, 13, 51, 20, 24, 48, 19, 60, 35, 66.

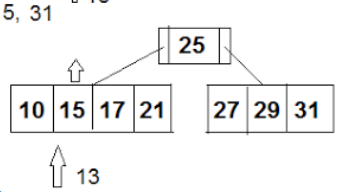
Entra : 10, 27, 29, 17. en orden



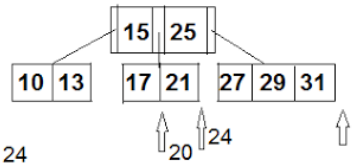
Entra: 25.



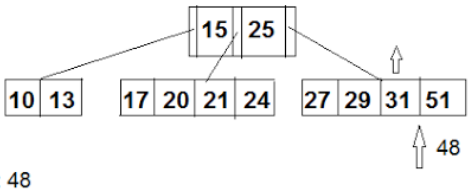
Entra: 21,15, 31



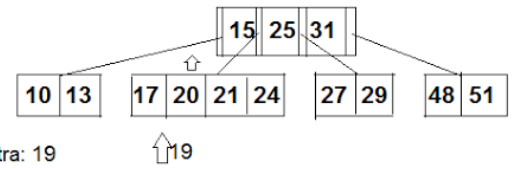
Entra: 13.



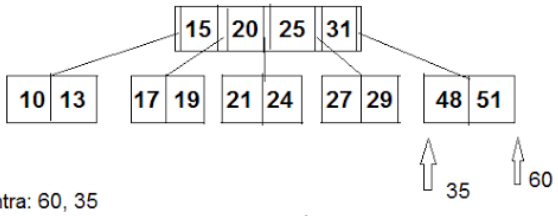
Entra: 51, 20, 24



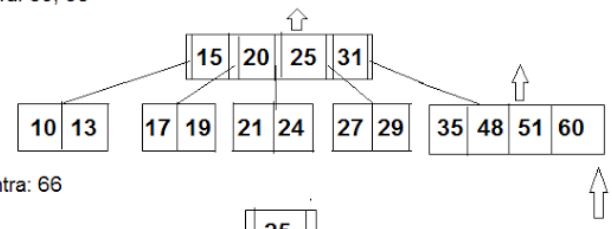
Entra: 48



Entra: 19



Entra: 60, 35



Entra: 66